

Prospective Data Model and Distributed Query Processing for Mobile Sensing Data Streams

Mariem Brahem, Karine Zeitouni, Laurent Yeh, and Hafsa El Hafyani

DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines
Paris Saclay University, France
{`mariem.brahem,karine.zeitouni,laurent.yeh,`
`hafsa.el-hafyani`}@uvsq.fr

Abstract. With the rapid advancements of sensor technologies and mobile computing, Mobile Crowd-Sensing (MCS) has emerged as a new paradigm to collect massive-scale rich trajectory data. Nomadic sensors empower people and objects with the capability of reporting and sharing observations on their state, their behavior and/or their surrounding environments. Processing and analyzing this continuously growing data raise several challenges due not only to their volume, their velocity, and their complexity but also to the gap between raw data samples and the desired application view in terms of correlation between observations and in terms of granularity. In this paper, we put forward a proposal that offers an abstract view of any spatio-temporal data series as well as their manipulation. Our approach allows to support this high-level logical view and provides efficient processing by mapping both the representation and the manipulation to an internal physical model. We explore an implementation within a distributed framework and envision the adaptation of data organization methods combining aggressive indexing and partitioning over time and space. The mapping from the logical view and the actual data storage will lead to revisiting the traditional database query rewriting and optimization techniques. This proposal is a first step in the objective of coping with the complexity, the imperfection of large data sizes in the MCS context.

Keywords: Spatio-temporal data modeling · Mobile crowd-sensing · Query processing.

1 Introduction

The recent advances in sensing technologies and mobile computing have paved the way for the emergence of the Mobile Crowd-Sensing (MCS) [18, 19] concept, leading to a continuous generation of large volume of rich trajectory data. More and more people rely on mobile devices (e.g., smartphone, tablets ...) and wearable sensors to share observations on their state, their behavior and/or their surrounding environments such as noise level, temperature or pollution conditions.

The growing scale of sensed data (Volume) coupled with real time sensor observations (Velocity) requires an effective model and efficient processing of complex spatio-temporal queries. In the past, processing large-scale data or high velocity data has been a bottleneck. Recently, big data analytics systems are becoming a de facto standard in massive data handling. While such systems fit well the large scale nature of sensed data, several issues related to the gap between raw data samples and the desired application view in term of correlation between observations occurring in different locations, or between different periods of time and spatial/temporal granularity (Variety) are still open. The heterogeneity and the diversity of sensor handsets from different manufacturers (with different sensitivities, time resolutions, and noise immunity) necessitate both an abstract data model, and an efficient implementation and analytics mechanisms. Currently, existing approaches mainly address historical spatio-temporal data to deal with the volumetry aspect [4, 1, 2] or stream time series to deal with continuous queries [15, 7, 3]. While these systems are efficient for batch or stream time series, there is a lack of a unified approach that combines batch and stream processing and tackles the unique characteristics of spatial mobile sensing data streams modeling and processing. In this paper, we present a prospective data model and a query processing module for sensed data streams. Our approach offers a high-level logical view of **Spatio-Temporal Data Series (STDS)** as well as an internal physical model that combines aggressive indexing and partitioning over time and space to dissolve the heterogeneity and the variety of data. We introduce an incremental query processing approach within a distributed framework to take into account the real-time processing of continuous queries, the large volume, and the high velocity of data. Our contributions are as follows:

- A high level logical view of STDS and a multi-granular physical data model that combines temporal and spatial partitioning.
- An extension of a unified distributed framework for big stored and stream data.
- A query optimizer within an incremental query processing model that offers a set of customized transformations rules for the optimization of spatio-temporal queries.

The rest of this paper is organized as follows. Section 2 discusses the major challenges related to big sensor data. Section 3 presents the related work while Section 4 provides an overview of our system architecture. In Section 5, we explain the details of the data model. The query processing workflow is presented in Section 6, and Section 7 summarizes the paper and provides some directions for future works.

2 Challenges of STDS Management

Data measured by mobile sensors can be represented by multivariate time series with a focus on the spatial dimension in addition to the temporal one. Such trajectory data denote the paths traced by sensors moving in space over time. Besides measuring the series of geographical positions over time, trajectory data

may also contain additional time-dependent variables such as the measurements of surrounding air pollution of the moving object. This large volume of data exhibits a number of challenging characteristics:

Spatial and Temporal Autocorrelation. From the modeling view, a distinctive aspect of such data series is the spatial autocorrelation, meaning that close objects tend to be more similar than distant objects. The same holds for consecutive observations on the same device. As a result, collected data from moving objects cannot be modeled as independent data, and specific algorithms taking into account the correlation between observations occurring in different locations, or between different periods of time need to be considered.

Data Heterogeneity. A notable characteristic is the heterogeneity in space and time. The strength of MCS relies on the usage of different types of sensors designed by different manufacturers that may vary in their sensitivity, sampling frequency, and noise immunity. The data collected from all sensing object should be merged, which could lead to measurements at irregular time intervals and missing data problems. We could observe timestamps that are closely spaced or too sparse in different cases. In fact, some sensors may be offline for hours or stay idle when the device is static (some sensors use the accelerometer to control the sampling rate), they can switch to a burst mode in some situations (increasing the sample frequency more than the normal rate) or stop transmitting the data if the variation is less than a predefined threshold, we could also get different sensor position resolutions. Such heterogeneous data sources should be taken into account in the model, and a harmonized view on the data is highly desirable in order to facilitate their processing and analytics.

Multi-Granularity. Besides, one of the most fundamental characteristics of mobile sensor data is the diversity of their granularity, both under the temporal and spatial dimensions. The temporal domain is typically represented at different time granularities. The spatial entity can be represented using a hierarchical representation that describes the subdivision of the spatial domain into different regions or cells. Combining multiple datasets with several granularities or changing the granularity of a dataset are important analysis task that we intend to deal with. Thus, we need to define a multi-granularity framework that takes into account the definition of the spatial and temporal granularities.

Data Volume. Huge amounts of data are being collected continuously from ubiquitous sensor-enhanced mobile devices (as many as the number of equipped holders) in different geographical areas. This requires leveraging big data processing techniques (e.g., Hadoop or Spark) to achieve in-depth understanding, and provide useful information.

Data Velocity. New rows in STDS are typically inserted in recent time intervals as appended rows. Thus, it is necessary to maintain efficient storage structures to handle the velocity of newly arriving data. The commonly used technique in online systems is to consider recent data as more relevant and flush old data. The limitation of such an approach is that some historical data is deleted, as a result, it misses the opportunity to process such data.

Continuous Queries. Due to the continuous processing of sensor data, spatio-temporal queries should be evaluated continuously, which necessitates an incremental processing paradigm. Traditional approaches for processing spatio-temporal data rely on historical data. While analyzing such archived data is important, it lacks the real-time processing of continuous queries. We need a platform that integrates a range of big data technologies to combine the processing of historical and real-time data. A new system architecture that handles massive volume of spatio-temporal data, covers the unique characteristics of sensor data and integrates batch and dynamic processing is necessary.

3 Related Work

Nowadays, sensor data processing can be oriented towards two perspectives: either an offline approach for querying historical data or an online approach for real time queries.

3.1 Offline Processing of STDS

Considerable research efforts have been devoted to offline management and analysis of big trajectory data (multi-dimensional time series) [17, 16, 35, 29]. These works are characterized by a complete storage of large historical data. Such data is used for offline analysis and knowledge discovery. Depending on the application type and the queries, the system tries to optimize query processing over the entire data. The key idea is to use a partitioning mechanism and distribute query processing among multiple nodes using distributed systems, such as Hadoop or Spark. Most of current works are oriented towards exploiting spatial indexes to design efficient methods for optimized query processing while preserving spatial locality. The objective is to tune the system and optimize spatio-temporal queries by making the best use of existing spatial and temporal indexes. There are three approaches for indexing trajectory data. The first approach is to consider the time dimension as the first dimension besides the spatial location. It divides the time dimension into multiple intervals and builds a spatial index (e.g., R-tree) for the trajectories in each time interval [6], or partitions spatial data within each time interval into spatial chunks and loads only relevant chunks for processing [4]. The second approach is to avoid discrimination between the spatial and the temporal dimensions using 3D space-filling curves techniques as proposed in Geomesa [2]. This approach allows to map spatio-temporal points into a single dimension and ensure data locality. It is efficient for queries that combine both temporal and spatial criteria. This category includes also the variations and extensions of R-trees: 3D R-tree, TB-tree, STR-tree [27]. The third approach is to alternate time and space. For example, T-PARINET [28] is based on a combination of spatial partitioning and B+-tree local indexes. Except T-PARINET, the aim of such systems is indexing large historical trajectory data. Therefore, they remain limited when it comes to the support of real time application which central goal is to minimize update costs and to support continuous queries. Moreover, they do not encompass other dimensions than space and time, as required in MCS context.

Another related area of research is distributed time series management systems, a survey on existing systems can be found in [21]. Flint [1] is a time series library for Apache Spark, it proposes interesting features for time series manipulation (temporal join, aggregation, moving average ...). It builds a time series aware data structure (*TimeSeriesRDD*) that allows to associate a time range to each partition and preserves the temporal order of data. Flint does not consider the spatial dimension and the real time nature of queries. It also lacks optimization techniques for temporal queries. However, in real life applications, data collected from sensors is represented by multi-dimensional time series where one dimension corresponds to the spatial location traversed by moving objects over time. TimeScaleDB [4] extends PostgreSQL query planner, data model, and execution engine to support SQL queries on time series data. Internally, it splits tables into chunks, each chunk corresponds to a specific time interval and a region of the partition keys space (using hashing). The created partitions are disjoint, which allows the query planner to select only required chunks to resolve a query. While apt at scaling SQL queries on large volume of data, this system is not designed for real time operations, as it lacks the ability to handle continuous queries and stream processing.

3.2 Online Processing of STDS

There are several commercial solutions for stream processing such as Samza [3], Flink [15], Spark Structured Streaming [7, 33], Storm [31]. While these systems generally support high ingestion rate and continuous queries, they are not designed for spatial time series. Academic architectures [5, 32, 23] were proposed in the literature focusing on streaming data and ignoring historical data. The PLACE [25] server is a data stream management system that supports continuous query processing of spatio-temporal streams. It employs an incremental evaluation paradigm that allows to continuously update the query answer and proposes high-level algorithms for continuous spatio-temporal queries. Zhang et al [34] extends Apache Storm to process data streams for moving objects, they employ a distributed spatial index to process continuous queries (e.g., continuous kNN). SCUBA [26] allows continuous spatio-temporal queries on moving objects. It proposes clustering techniques to group moving objects and queries into moving clusters based on common spatio-temporal properties to optimize query execution. However, the massive volume of historical trajectory data have exceeded the capacities of such streaming architectures. Management and indexing aspects of large volume of data were not their main concern. Batch processing is still needed for data analysis on historical data. Besides, for some queries, all the data is necessary to ensure a more accurate query result. As a result, a common strategy is to use a hybrid architecture that combines stream and batch processing.

3.3 Unified Approach for STDS Management

The growing volume of spatial data series and the rapid increase of the velocity of sensor data streams accelerate the need for a big data architecture that of-

fers continuous processing of data streams. With the emergence of the volume and the velocity issues, Marz introduced the *Lambda architecture* [24] to handle query processing in a scalable and a fault-tolerant way. It is composed of three layers: the batch layer, serving layer, and speed layer. The batch layer is responsible for processing historical data for batch analysis. The speed layer focuses on analyzing incoming streaming data in near real-time and the serving layer aims at merging the results from the previous two layers. This architecture allows to handle large-scale data and integrate batch and real-time processing within a single framework. While the *Lambda architecture* achieves its goal, it comes with high complexity and redundancy. Kreps [22] discussed the disadvantages of the *Lambda architecture* and presented a new approach for real time data processing named the *Kappa architecture*. This architecture favors simplicity by merging the batch and streaming layers and avoiding data replication. Inspired by the *Kappa architecture*, Spark Structured Streaming [7] is a streaming computation system that combines batch and stream processing using the same code. Based on the *Lambda architecture*, PlanetSense [30] is a generic platform for gathering geospatial intelligence from real time data (e.g., social media, passive and participatory sensing). It combines the power of archived data and the dynamics of real time data for spatio-temporal analytics. However, such an architecture inherits the limits of the *Lambda architecture*, as it needs to implement the transformation logic twice, once in the batch system and once in the stream processing system. In contrast to our proposal, PlanetSense does not propose to manipulate temporal operations specific to time series and lacks a representative data model for time series data taking into account the heterogeneity of data.

4 System Overview

Figure 1 describes our vision of a unified framework for processing batch and streaming STDS.

Data Sources. There are two types of data ingestion in the system. The first is batch data ingestion (finite datasets) that consists of loading a previously acquired data (e.g., loading the data from a previous campaign as a cold start of the data store). The second type corresponds to streaming data (infinite datasets) collected from current campaigns. Notice that we archive the acquired data for further batch analysis, when needed.

Core Engine. The main data processing is performed within the core engine. It contains three components: the query parser, the optimizer and the indexing component (as detailed hereafter). Internally, we rely on existing distributed and streaming engines to generate parallel dataflows. Our target is not a pure streaming framework that requires hard real-time handling. Therefore, we have chosen Spark Structured Streaming as a backend and exploit the parallel receivers of Spark Streaming to collect data from different data sources. Our queries are processed using the Spark’s micro-batch model, which processes data streams as a series of small batch jobs.

Query Parser. The query parser is responsible for checking and validating the query syntax. It translates an algebraic expression into a set of transfor-

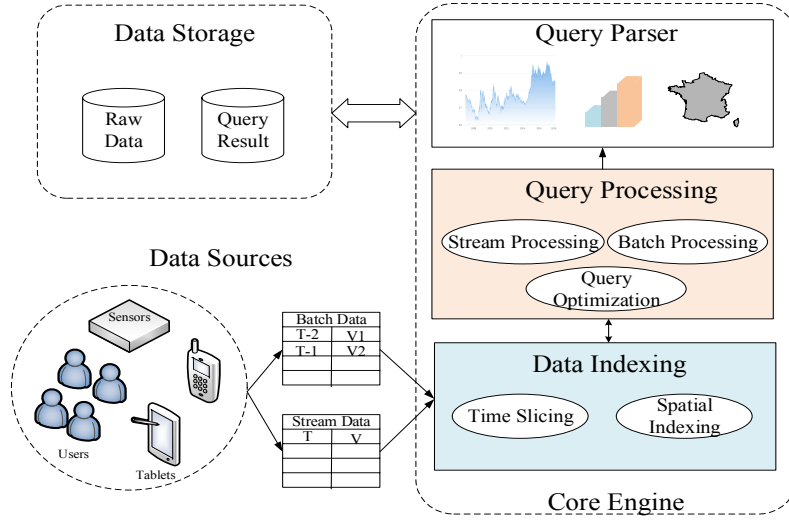


Fig. 1: System Architecture

mations on Spark Streaming DataFrame, such as *selection*, *projection*, *temporal join*, *shift*, *aggregation* to convert values to coarser or finer granularity. These operators form an algebra, as an extension of the relational algebra to account for the semantic of STDS (see Section 5 for more details). Subsequently, we propose to extend the Spark Streaming DataFrame API to support spatio-temporal operations.

Query Processing. Our query optimizer creates a series of incremental execution plans from a streaming logical plan. The idea is to inject appropriate optimization rules to obtain optimized execution plans. Rule-based optimization in our context exploits spatial indexing and time slicing to access the smallest possible number of partitions, avoiding cartesian product (in case of temporal or spatial joins) or performing selection on required time intervals as early as possible ... Indeed, our query optimizer injects optimization rules to avoid scanning all the data series and eliminates records that do not contribute to the query result. Our query processing module uses the Spark streaming DataFrames/Datasets API to combine batch and stream processing. This data structure allows to represent bounded data as well as streaming data. The advantage is that we can apply the same operations on static and streaming DataFrames.

Indexing. We integrate the concept of indexing to achieve better query performances. We employ time slicing to divide the input data into multiple slices that are distributed on their time granularity. Data within each slice is further divided into sub-slices based on spatial indexing techniques. The key idea is to consider the temporal dimension first, which is important for time series analysis. Details about our physical model are presented in Section 5.3.

Data Storage. In order to provide accurate and real-time analysis, we maintain different data storage for raw data and query (continuous) results. Raw data refers to incoming stream time series data persisted for batch analysis using *Parquet* format. New streaming data are maintained in memory until it exceeds a memory threshold (specified by the user). Once the threshold is reached, it is flushed to a separate data storage. This is done periodically using an *append mode* to the DataFrame. Doing so, data are blocked in output files which size is controlled, avoiding the inefficient multiplicity of small files resulting from the default mode.

5 Data Model

As a running example, we consider a database derived from Polluscope¹. A cohort of volunteers is collecting sensory data each in a different STDS, possibly at different granularities. For instance, GPS data are acquired at a higher frequency than air pollutants such as PM2.5, NO2, and Black Carbon.

5.1 Preliminaries

The notion of granularities has been deeply studied in the literature, Bettini et al [9, 10, 8] define the temporal granularity as a partition of the time domain.

Definition 1. (*Temporal Granularity*). Formally, a temporal granularity g_T is a function from an ordered set I_T to the power set of the temporal domain T such that:

$$\begin{aligned} \forall i, j, k \in I_T, (i < k < j \wedge g_T(i) \neq \emptyset \wedge g_T(j) \neq \emptyset \implies g_T(k) \neq \emptyset) \\ \forall i, j \in I_T, (i < j \implies \forall x \in g_T(i) \forall y \in g_T(j) x < y) \end{aligned}$$

Typical examples of temporal granularities are days, weeks, months. $g_T(i)$ are called temporal granules of the granularity g_T . The first condition states that the subset of the set that maps to non-empty subsets of the time domain is contiguous. The second condition states that granules do not overlap and that their order is the same as their time domain order. Besides, Camossi et al. [14] define the spatial granularity as a mapping from an index set to subsets of the spatial domain (i.e. a set of 2-dimensional points)

Definition 2. (*Spatial Granularity*) Formally, a spatial granularity g_S is a function from an ordered set I_S to the power set of the spatial domain S such that:

$$\forall i, j \in I_S, (i \neq j \wedge g_S(i) \neq \emptyset \wedge g_S(j) \neq \emptyset \implies \text{intersects}(g_S(i), g_S(j)) \neq \emptyset)$$

Typical examples of spatial granularities are pixels of different sizes, or a spatial hierarchy such as administrative subdivisions of a country. $g_S(i)$ are called spatial granules of the granularity g_S .

¹ A French project to build a participative observatory for the surveillance of individual exposure to air pollution and health effects. ANR-15-CE22-0018 Grant. <http://polluscope.uvsq.fr>

5.2 Logical Data Model

Definition 3. (*Time Series*). We define a time series as an infinite sequence of values where a value is a couple (t, v) where $t \in T$ is a timestamp (at a given granularity) from a time domain T with discrete time units in increasing order and v is a vector (v_1, \dots, v_n) where each value is a measurement or scalar value, v is an n -tuple of a fixed size.

The left side Figure 2 shows an example of a time series that records the time t and the corresponding values (e.g., PM2.5, PM10, NO2) captured in a vector v .

Definition 4. (*Spatio-Temporal Data Series*). We define a Spatio-Temporal Data Series (STDS) as a time series where the location (e.g., latitude and longitude) belongs to the vector v .

Definition 5. (*Empty Value*). The empty value (denoted '!') means that there exist no value. As a time series R is an infinite sequence of vectors, $\forall v \in R, \text{time}(v) \in T$. If v is not defined at a given time, then $v = !$.

Definition 6. (*Unknown Value*). The unknown value (denoted '??') means that the value is undefined. It is equivalent to the NULL value in the relational model.

Using our model, a time series constitutes a linear space vector (a collection of vectors). This mathematical structure allows us to apply basic vector space operations (*plus, minus, scale*). Thus, time series could be added (denoted $+$), multiplied (denoted $*$) by numbers (scalar) or even combined linearly in expressions such as $TS1 + s * TS2$ (s being a real). Our model includes also an extension of the relational algebra operators to the support of time series. To this end, we revisit the operators such as temporal *selection, projection, join, union, intersection, aggregation* as follows.

Definition 7. (*Temporal Selection*). The temporal selection applied to a time series is a time series where we replace the original value with an empty value (!) if the predicate is not satisfied. Denoting (t, v) the entry t of vector v of the processed time series:

$$TSEL_{pred}(R) = \{(t, v) \mid (t, v) \in R \wedge pred(v)\} \cup \{(t, !) \mid (t, v) \in R \wedge !pred(v)\}$$

Example: Selection of air pollutants exceeding a certain threshold (e.g., 50).

$$TSEL_{vi>50}(R)$$

Definition 8. (*Window Selection*). We define a window selection operator as a transformation that selects values satisfying a temporal predicate w . Formally:

$$WSEL_w(R) = \{(t, v) \mid (t, v) \in R \wedge overlaps(t, w)\}$$

Example: Selection of air pollutants during a specific time period.

$$WSEL_{[01/06/2019,15/06/2019]}(R)$$

Definition 9. (*Temporal Projection*) We define a temporal projection operator as a transformation that applies a function to each value of the time series it is applied on. Formally:

$$TProj_f(R) = \{(t, (v_1, \dots, v_k)) \mid (t, v) \in R \wedge v_i = f(v) \text{ for } i \in \{j_1, j_2, \dots, j_k\}\}$$

Where f is a linear function that preserves vector addition and scalar multiplication.

Example: Projection and multiplication of PM10 values by 2.

$$TProj_{v2*2}(R)$$

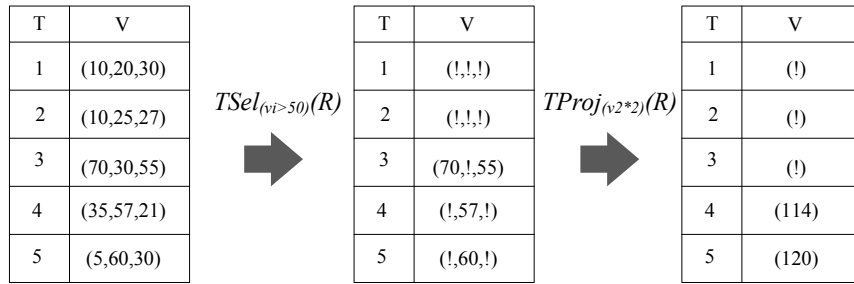


Fig. 2: Selection and Projection with Examples

Definition 10. (*Shift*) We define a shift operator as a transformation that applies a shift (future or past) to each timestamp of the time series. Formally:

$$Shift_\delta(R) = \{(t', v) \mid (t, v) \in R \wedge t' = t + \delta\}$$

Example: Shift the environmental time series by 1 day in order to compare the exposure to pollutants between two consecutive days.

$$Shift_{1day}(R)$$

Definition 11. (*Temporal Intersection, Difference & Union*). We define an adaptation of the relational intersection, difference and outer union as follows:

$$S_1 \cap S_2 = \{(t, v) \mid (t_1, v_1) \in S_1 \wedge (t_2, v_2) \in S_2 \wedge t = intersects(t_1, t_2) \wedge v = v_1 = v_2\}$$

$$S_1 - S_2 = \{(t, v) \mid (t, v) \in S_1 \wedge (t, v) \notin (S_1 \cap S_2)\}$$

$$S_1 \cup S_2 = \{(t, v) \mid (t, v) \in (S_1 - S_2) \vee (t, v) \in (S_2 - S_1) \vee (t, v) \in (S_1 \cap S_2)\}$$

Definition 12. (*Temporal Aggregation*). Let R be a time series with a timestamp attribute t , f be an aggregation function (e.g., sum, count, avg) that takes a set of values as an argument and applies an aggregation. We define a set G_T that contains all granules τ of granularity g_T such that $G_T(R) = \{\tau \mid \tau \in$

$cast(t, g_T) \wedge t \in R\}$. Each element of G_T defines a partition $S(\tau, R)$ of R such that:

$$S(\tau, R) = \{v \mid (t, v) \in R \wedge overlaps(time(v), \tau)\} \quad (2)$$

The temporal aggregation is defined as:

$$TAgg_{(g_T, f)}(R) = \{(\tau, f(S(\tau, R))) \mid \tau \in G_T(R)\}$$

The set G_T (e.g., $G_T = \{\text{March, April, June, July}\}$) ranges over the granules of a granularity g_T (e.g., months). $S(\tau, R)$ collects all the values $v \in R$ such that $time(v)$ is overlapping τ . A result tuple is then produced by extending τ with the result of the aggregate function f that is computed over each element of $S(\tau, R)$. Example: Compute the monthly average of all air pollutants.

$$TAgg_{(months, avg)}(R)$$

Definition 13. *Window Aggregation.* This operator allows to partition and aggregate values over time, based on a moving window w . We define a set $W_T(R) = \{\tau_1, \tau_2, \tau_3, \dots, \tau_q\}$ that contains the collection of time intervals dividing the time horizon t_I of R into sub-intervals of the size of the window w such that $t_I = \bigcup_{i=1}^q \tau_i$. Each element of W_T define a partition $S(\tau, R)$ as defined in Equation 2. Thus, we define the window aggregation as:

$$WAgg_{(w, f)}(R) = \{(\tau, f(S(\tau, R))) \mid \tau \in W_T(R)\}$$

Example: Compute the average over a half an hour moving window of a specific air pollutant.

$$WAgg_{(30min, avg)}(R)$$

Definition 14. *(Temporal Join)* The temporal join between two time series R and S allows to append each row in R with the row in S at the same time values. Formally:

$$TJoin(R, S) = \{(t, v) \mid (t_1, v_1) \in R \wedge (t_2, v_2) \in S \wedge v = v_1 \oplus v_2 \\ \wedge t = intersects(t_1, t_2) \wedge t \neq \emptyset\}$$

Example: Match the environmental time series (R) with the GPS data (L).

$$TJoin(R, L)$$

Definition 15. *(Shift Temporal Join)* The future (past) temporal join between two time series R and S allows to append each row in R with the closest future row in S at or after (before) a time interval δ . It is a redefinition of the temporal join and the shift operators. Formally:

$$TJoin_{\delta}(R, S) = \{(t, v) \mid (t_1, v_1) \in R \wedge (t_2, v_2) \in S \wedge v = v_1 \oplus v_2 \\ \wedge t = intersects(t_1 + \delta, t_2) \wedge t \neq \emptyset\}$$

Example: Match the environmental time series (R) with the shifted humidity (H) which variation may impact the values in R after 3mn.

$$TJoin_{3mn}(R, H)$$

Definition 16. (*Spatial Aggregation*) We define the spatial counterpart of G_T denoted G_S such that $G_S(R) = \{s | s \in \text{cast}(\text{location}(t), g_S) \wedge t \in R\}$ where g_S is a spatial granularity and $\text{location}(t)$ gives the spatial element of a timestamp $t \in R$. Each element of G_S defines a subset of values as follows:

$$S_{g_S}(s, R) = \{v | (t, v) \in R \wedge \text{overlaps}(\text{location}(t), s)\}$$

Similar to the temporal aggregation, the spatial aggregation is defined by the following expression:

$$SAgg_{(g_S, f)}(R) = \{(s, f(S_{g_S}(s, R))) | s \in G_S(R)\}$$

Example: For each country, compute the highest value of a specific air pollutant.

$$SAgg_{(\text{countries}, \text{max})}(R)$$

We could also propose other operators such as a *split* operator which subdivides a single time series into multiple segments, or a *clustering* that partitions a time series according to consecutive similar values, or spatio-temporal join by adding a spatial predicate.

5.3 Physical Data Model

Physically, an infinite sequence of values cannot be stored. As in [20], at storage level, we propose a discrete model of time series data to implement the infinite sequence as in the logical (abstract) model. We physically encode time series data as a set of sequences with specific metadata. As streaming time series enters our framework, each timestamp is represented by a positive integer named Epoch and a granularity. The granularity is chosen based on the original precision of data (e.g., 2019/06 corresponds to a one-month granularity).

To reduce the storage overhead, for each sequence of a time series, we only store an array of values omitting the actual timestamp. In fact, we associate a start Epoch value s , and a granularity g as metadata. Then the timestamp of the i^{th} values of the array can be recovered by the simple formula $s + i * g$. If the analysis needs to operate over raw time format, then we use the metadata to calculate the row number of the corresponding values. Our model allows to manage missing data, for example, if there is no value between two sequences, at the storage level nothing is stored, at the logical level, it is represented by an empty value denoted as '!'. To summarize the first phase of our proposed physical model, time series data is organized as sequences of lists where each list is represented by a granularity. This allows to structure data in a temporal hierarchy where each granularity is represented as a level in the hierarchy.

In order to speed-up the access and the filtering on time series, and in particular to STDS data, we propose a model that alternates temporal and spatial indexing. We envision a two-level partitioning scheme, where the first level follows a global index, and the second depends on a second index. At the lowest level, the data will be further divided into even smaller spatial sub-partitions

called *buckets*. For instance, the first level can leverage a spatial index while the second follows a temporal partition, and the *buckets* could be based on the order of spatial filling curves. The way the spatial and temporal dimensions will be alternated is not yet decided and may require fine tuning to adapt to the data and the query profile. For query processing, our system will only scan the content of the spatial *bucket* (e.g., `/sid=20`) that can be accessed from the temporal partition (e.g., `/ts.parquet/nump=10`) that contains a range of Epoch indices. This physical organization is inspired by the physical optimization employed for large volume of astronomical data [13, 11]. It has proved its efficiency, as it processes the query in a way that makes it efficient to retrieve the contents of required spatial buckets, obviate scanning irrelevant partitions and allow fast aggregation queries for granularity conversions.

We use an *append* save-mode to load additional data while avoiding to overwrite existing data. Data is archived in time-ordered partitions. New incoming time series naturally arrives time-ordered, this allows new data to be appended to existing partitions rather than having to re-sort data into previously stored partitions.

6 Query Processing

Spark Structured Streaming is based on the Spark Catalyst extensible optimizer [7] which allows adding new optimization techniques. Our knowledge of the Spark Catalyst optimizer helped us in designing a new query processing model for STDS. Figure 3 represents our query processing workflow that consists of three major steps: Extended Analysis, Incrementalization and Extended logical-physical optimizations. The input is an algebraic expression that is translated into a set of transformations on Spark Streaming DataFrames by the query parser. This allows to leverage the optimizations offered by the Catalyst optimizer and to inject new optimization techniques.

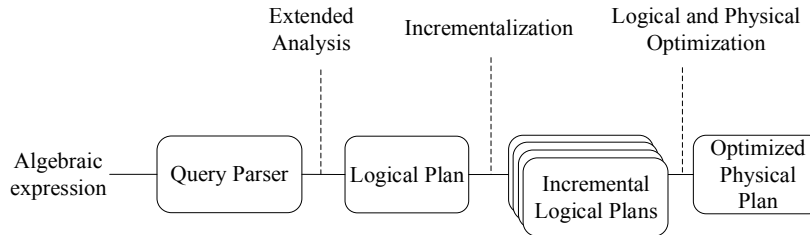


Fig. 3: Query Processing Workflow

Extended Analysis. The Extended Analysis extends the Spark Structured Streaming analysis to resolve spatio-temporal operations. It validates the query and resolves the attributes and data types. We intend to extend the catalyst

optimizer to inject resolution rules in order to transform an impossible-to-solve plan into an analyzed logical plan.

Incrementalization. The next step is to use the Spark Structured Streaming incrementalization technique that allows to continuously update results in response to new data. The main idea is to only report the changes in the query result since the last trigger. This ensures a fast query evaluation because we limit the amount of data to get the query result. We rely on the concept of incremental algorithms to transform queries into trees of traditional logical operators (e.g., join, filter ...).

Extended logical and physical optimizations. The objective of this step is to exploit logical optimization to transform the logical plan into an optimized logical-physical plan using indexing and partitioning. It allows to solve the filtering phase of some proposed queries by either transforming the temporal join query into an equi-join or by filtering the relevant partitions required by spatial or temporal ranges... Our query processing module applies multiple optimization rules to map a query plan to a semantically equivalent plan. Some examples of optimization rules that could be included:

- Temporal Partition Pruning. It consists in determining the temporal partitions that need to be scanned, and hence the epoch indices that can be pruned, to get the query result. Such a rule could be applied for selection queries and allows to access relevant values using their rows number.
- Spatial Index PushDown. This optimization allows to inject new filters in the query plan to eliminate loading spatial objects that do not contribute to the query result. This ensures that such filters are applied at the low level rather than dealing with the entire temporal partition.
- Avoiding Cartesian Product. Temporal join queries can be conceptually formulated as cartesian products. A possible optimization is to replace this product by an equi-join on Epoch indices. The trick is to take inspiration from our spatial join algorithm [12] by shifting all objects of a reference dataset on the fly and transforming the start epoch values. Then, a simple equi-join query on epoch indices becomes sufficient to generate candidate results.

7 Conclusion

In this paper, we presented a unified framework for mobile sensing big stored and stream data. Our framework extends Spark Structured Streaming with the adaptation of data organization and the injection of various optimization rules to optimize processing of stream and historical data series. We also presented a logical data model for STDS and a multi-granular internal data model to take into account the heterogeneity of data. We presented the key query processing workflow of our framework to support incremental algorithms and logical/physical optimizations. Currently, we are working on an integrated prototype within Spark and Catalyst to support an important application branch of MCS which is air quality sensing where air pollution is monitored using multi-sensor devices within the Polluscope project.

Acknowledgements

This work benefited from the support of the project POLLUSCOPE ANR-15-CE22-0018 of the French National Research Agency (ANR). It has been also supported by the MASTER project that has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie-Slodowska Curie grant agreement N. 777695.

References

1. Flint. <https://github.com/twosigma/flint> (Last accessed Mai 2019)
2. Geomesa. <https://www.geomesa.org/> (Last accessed Mai 2019)
3. Samza. <http://samza.apache.org/> (Last accessed Mai 2019)
4. TimeScale. <https://www.timescale.com/> (Last accessed Mai 2019)
5. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Conway, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: a new model and architecture for data stream management. *The VLDB Journal* **12**(2) (2003)
6. Alarabi, L., Mokbel, M.F., Musleh, M.: St-hadoop: A mapreduce framework for spatio-temporal data. *GeoInformatica* **22**(4) (2018)
7. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., Zaharia, M.: Structured streaming: A declarative api for real-time applications in apache spark. In: *Proceedings of the International Conference on Management of Data* (2018)
8. Bettini, C., Jajodia, S., Wang, S.: Time granularities in databases, data mining, and temporal reasoning. Springer Science & Business Media (2000)
9. Bettini, C., Wang, X.S., Jajodia, S.: A general framework for time granularity and its application to temporal reasoning. *Annals of mathematics and artificial intelligence* **22**(1-2) (1998)
10. Bettini, C., Wang, X.S., Jajodia, S., Lin, J.L.: Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering* **10**(2) (1998)
11. Brahem, M., Yeh, L., Zeitouni, K.: Efficient astronomical query processing using spark. In: *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2018)
12. Brahem, M., Zeitouni, K., Yeh, L.: Hx-match: In-memory cross-matching algorithm for astronomical big data. In: *International Symposium on Spatial and Temporal Databases* (2017)
13. Brahem, M., Zeitouni, K., Yeh, L.: Astroide: A unified astronomical big data processing engine over spark. *IEEE Transactions on Big Data* (2018)
14. Camossi, E., Bertolotto, M., Bertino, E.: A multigranular object-oriented framework supporting spatio-temporal granularity conversions. *International Journal of Geographical Information Science* **20**(05) (2006)
15. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **36**(4) (2015)
16. Ding, X., Chen, L., Gao, Y., Jensen, C.S., Bao, H.: Ultraman: A unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment* **11**(7) (2018)

17. Fang, Y., Cheng, R., Tang, W., Maniu, S., Yang, X.: Scalable algorithms for nearest-neighbor joins on big trajectory data. *IEEE Transactions on Knowledge and Data Engineering* **28**(3) (2015)
18. Ganti, R.K., Ye, F., Lei, H.: Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine* **49**(11) (2011)
19. Guo, B., Wang, Z., Yu, Z., Wang, Y., Yen, N.Y., Huang, R., Zhou, X.: Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys (CSUR)* **48**(1) (2015)
20. Güting, R.H., Behr, T., Düntgen, C., et al.: Secondo: A platform for moving objects database research and for publishing and integrating research implementations. *IEEE Data Eng. Bull.* **33**(2) (2010)
21. Jensen, S.K., Pedersen, T.B., Thomsen, C.: Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering* **29**(11) (2017)
22. Kreps, J.: Questioning the lambda architecture. Online article, July (2014)
23. Marz, N., Warren, J.: *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co. (2015)
24. Marz, N., Warren, J.: *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co. (2015)
25. Mokbel, M.F., Xiong, X., Aref, W.G., Hambrusch, S.E., Prabhakar, S., Hammad, M.A.: Place: a query processor for handling real-time spatio-temporal data streams. In: *Proceedings of the international conference on Very large data bases-Volume 30* (2004)
26. Nehme, R.V., Rundensteiner, E.A.: Scuba: scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In: *International Conference on Extending Database Technology* (2006)
27. Pfoser, D., Jensen, C.S., Theodoridis, Y., et al.: Novel approaches to the indexing of moving object trajectories. In: *VLDB* (2000)
28. Sandu Popa, I., Zeitouni, K., Oria, V., Barth, D., Vial, S.: Indexing in-network trajectory flows. *The VLDB Journal* **20**(5) (2011)
29. Shang, Z., Li, G., Bao, Z.: Dita: Distributed in-memory trajectory analytics. In: *Proceedings of the 2018 International Conference on Management of Data* (2018)
30. Thakur, G.S., Bhaduri, B.L., Piburn, J.O., Sims, K.M., Stewart, R.N., Urban, M.L.: Planetsense: a real-time streaming and spatio-temporal analytics platform for gathering geo-spatial intelligence from open source data. In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2015)
31. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al.: Storm@ twitter. In: *SIGMOD international conference on Management of data* (2014)
32. Tran, D.A., Hua, K.A., Do, T.T., et al.: A peer-to-peer architecture for media streaming. *IEEE journal on Selected Areas in Communications* **22**(1) (2004)
33. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: Fault-tolerant streaming computation at scale. In: *Proceedings of the twenty-fourth ACM symposium on operating systems principles* (2013)
34. Zhang, F., Zheng, Y., Xu, D., Du, Z., Wang, Y., Liu, R., Ye, X.: Real-time spatial queries for moving objects using storm topology. *ISPRS International Journal of Geo-Information* **5**(10) (2016)
35. Zhang, Z., Jin, C., Mao, J., Yang, X., Zhou, A.: Trajspark: A scalable and efficient in-memory management system for big trajectory data. In: *Asia-Pacific Web (AP-Web) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data* (2017)